THE UNIVERSITY OF ALBERTA

DESCRIBING LINE DRAWINGS TO A MACHINE

by

© Linda Easton

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

Fall 1971

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled DESCRIBING LINE DRAWINGS TO A MACHINE submitted by Linda Easton in partial fulfilment of the requirements for the degree of Master of Science.

Date June 25th 1971

ABSTRACT

This thesis studies the possibility that some insight might be gained from the more formal methods of defining classes of pictures using picture grammars, which suggests ways of designing more practical software for a graphics system. A number of examples showing the use of linguistic techniques to describe classes of pictures are discussed. Finally, the possible uses of the various aspects of these techniques for improving graphics software are considered.

ACKNOWLEDGEMENTS

The author acknowledges her indebtedness to Dr. J. P. Penny for suggesting the problem considered in this thesis; for his advice, criticism and guidance during the period that this work was being done; and for his financial support.

Also my thanks to all the people who spent time reading my work and offering constructive advice; and special thanks to Geoff Ewing for his continued moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

> "Computer Graphics represents a new addition to the communication spectrum available to the designer. ... Through thoughtful use of computer graphics, designers and engineers should have a more complete understanding of the problem at hand and should be able to produce designs more efficiently, economically and objectively."
>
> (Fetter, 1968)

> "The purpose of graphics is to alter meaningfully and positively the dialogue between man and machine so that the working relationship between them will be more productive."
>
> (Cancro and Slotnick, 1968)

> "Wide-scale application of interactive computer graphics (ICG) is currently inhibited by two major difficulties: 1) Terminal time costs too much, 2) It takes too much effort and expertise to develop ICG programs."
>
> (Boehm et al, 1969)

It has been suggested (Fetter, 1968) that computer graphics can be a useful tool for designers in various fields. These include circuit and aircraft design and various planning problems such as campus planning (Deecker, 1970). Not only should the use of computer graphics aid the designers in finding an economical and feasible solution, but it may also help the designer to understand his problem more thoroughly. Unfortunately, what should be true is not always, in fact, the case. Graphic techniques, at present, have not been as widely exploited as they might be.

One of the problems under consideration by people in the field of computer graphics today is to develop a more general software for graphical displays. One of the reasons for this study is that, as has been pointed out by Boehm (Boehm et al, 1969), the user must at present be an expert programmer as well as knowledgeable in his field.

It seems possible that we may be at a crossroads in the development of computer graphics software. One road will lead us to more expensive, larger and more complicated systems which will require even more specialization for the designer to use them. The other road will lead to more general software which will allow the user of the design system with little programming skill to use the graphical display for solving his problems. Obviously, the second system is the one which we want to be able to build for the user. A possible approach, which may help us to build this more general system, is to study each part of a graphics system as a unit rather than to study (and build) the entire system as a whole. Some of the parts which we may want to study include the use of data structures, command languages, and methods of describing classes of pictures in which a designer is interested.

This approach of studying the parts of a graphics system will lead us to the essential question which we shall consider in this thesis: Is there any insight to be gained from the more formal methods of defining classes of pictures

using picture grammars[1] to suggest ways of designing more practical software for a graphics system?

In the second chapter, this question is considered in more detail. We will discuss the aims of building a system with more practical software, a system which exists now, and the requirements of the system we want. In each case, we will give particular consideration to the part of the system in which we are most interested (i.e., the way in which pictures are defined). The final section of the second chapter discusses a linguistic grammar[1] and the devices which may be used to extend this type of grammar to describe classes of pictures.

In the next three chapters, we consider a number of examples of the use of linguistic techniques[1] to describe classes of pictures. In Chapter 3, we consider two articles written by R. S. Ledley. The applications, as well as the method, are discussed in detail so that the reader may become familiar with the processes involved in constructing and using a picture grammar. The first article deals with the syntactic description of drawings of houses and various methods that may use this description for identifying a house. The second article discusses the recognition and analysis of biomedical pictures which have been described by certain syntactic rules.

---

1. See the discussion in Appendix I.

We consider, in Chapter 4, a hierarchical system developed by R. Narasimhan for pattern recognition and description, and also for picture generation. Two examples of this system have been considered: One for describing bubble chamber pictures and the other for generating English letters. In Chapter 5, we discuss a Picture Description Language developed by Miller and Shaw. This language is used to write productions of grammars to describe classes of pictures.

The last section in each of these three chapters discusses certain aspects of the examples: for example, the methods which are used to adapt a grammar for describing linear strings to a picture grammar. At this point, we do not try to point out any advantages or disadvantages of one method over another.

In Chapter 6, we give a description of a class of very simple pictures (rectangles) as it would be given in existing graphics software. Also, we try to point out the ways in which the description is explicit and the ways in which it is implicit.

Chapter 7 contains a definition of the class of rectangles using each of the methods discussed earlier (in Chapters 3, 4 and 5). For each method, we point out the differences between that description and the one given in Chapter 6. Finally, the aspects of each method which

may help us to develop improved graphics software are discussed.

Possible improvements which might be made to existing graphics software are discussed in Chapter 8.[1] Some of the areas considered include the definition of display primitives, the definition and display of problem primitives, and the explicit description of a given class of pictures using the problem primitives.

---

1. This discussion is not limited to the classes of pictures studied in Chapters 6 and 7.

CHAPTER 2 - GRAPHICS SYSTEMS

2.1 The Problem:

The overall objective in this work is to determine formal methods of describing line drawings to a machine. This is one part of a wider investigation aimed at developing a computer graphics system, not necessarily of practical value, but rather with at least some attempt at a formal basis. It is necessary to remember that the implementation of this system is desirable, and therefore any design should consider the features and limitations of the graphics hardware. The aims in building an actual system are:

(1) to provide a unified framework within which several research problems can be approached, such as the study of picture grammars and console command languages.

(2) to provide insights into the inter-relationships between the various parts of a total graphics system.

(3) to obtain, perhaps, ideas which might be adapted for the building of practical software.

2.2 The System:

In particular, the system being designed is aimed at a subclass of graphics problems - those problems in which the user is constructing a drawing on the CRT. If we consider an *ad hoc* system for a given problem (such as the Computer Aided Logic Design (CALD) system of B. V. Johnson, 1969), we see that the user communicates with the graphics system

by the use of a console command language. The result of the user giving a sequence of these commands is the development of a compound data structure[1] model. This model represents an object of which the operator sees the drawing developing on the CRT. The idea that model construction, rather than drawing construction, is the primary objective of a graphics system for drawing on the CRT[2] seems appropriate since there may be many pictorial representations of an object described by one model. For example, we may want to be able to see different views - elevation, plan, perspective - of a house. We need only one description of the house. Gray (1967) has said that in computer-aided design, there is "a requirement for a general system for building models, to which can be applied transformations and algorithmic procedures. One such transformation may be display generation." From the single model of the house, we could, by using different tranformations, generate the different views required.

---

1. "'Compound Data Structure' is a very general term and includes several distinct forms or types of structure, each of which has its own special properties which must be specified; in particular, how to build and how to access the elements of the structure. In the most general sense, a compound data structure is a collection of objects, each with some associated data, arranged in such a way that the relationship of one object to another is explicit." (Gray, 1967).

2. We neglect here the use of the CRT as a pure "drawing board" where the picture has no formal structure.

Now consider a system which will develop models for a particular "drawing construction" problem - say, for example, the CALD system for construction of logic circuit diagrams. A number of problem primitives have been defined using the display primitives of the graphics hardware. One example of these problem primitives is the NOT-gate, defined in terms of the hardware display primitives, lines, points and symbols. These primitives are explicitly defined by the use of the Block definition CALL'S in GRIDSUB.[1] The gates may be joined to form subcircuits and circuits since the operator, by giving a sequence of commands, may have the system develop any one of a set of circuit models and the corresponding diagram. Though explicit definition of a set of well-formed models is not made, there is undoubtedly such a set defined implicitly (or programmed in) by the system's designer. The system cannot generate any models other than the particular set for which it was developed. (We make the assumption that the system is without bugs, and will yield only those models and diagrams which its designer intended.)

In this, and probably many other applications, there is a fairly obvious hierarchy of picture classes, (discussed in Chapter 6 in more detail):

(1) display primitives - lines, points, symbols

---

1. "GRIDSUB (Huen et al, 1970) is a fairly conventional graphics software package which, like all such packages, includes picture definition facilities."

(2) problem primitives constructed from (1) - for example, logic circuit elements

(3) subcircuits constructed from (2) - for example, half-adder

(4) complete circuits constructed from (3) - for example, adder.

Graphics software provides a means for the user to specify (usually in a quite informal way) how classes of pictures at one level can be constructed from those at a lower level, though the user probably does not think in these terms. What we would like to find is:

(a) more formal methods of specifying display primitives

(b) more formal methods of defining and displaying problem primitives (2 from 1)

(c) more formal methods of overall organization (3 from 2, 4 from 3 and 2).

Essentially, our aim for the full system (of which a picture definition facility is one part) is a general system rather than one designed for a particular "drawing construction" problem. We would like to be able to supply it with a description, as formal as possible, of all well-formed models and well-formed diagrams for a given problem. The system must be able to generate all such models and diagrams, and only those models and diagrams. The one(s) which it would generate during any terminal session would be that(those) requested by the operator.

Therefore, for this more general system a set of models could be defined explicitly at any time, and the set of models would be that for a particular problem.

2.3 Requirements of the System:

There are four requirements for building this system. The first is a formalism for defining the designer's command language. It is desirable that this formalism be easy to use as well as flexible. Since any "statement" in this language would probably consist of a linear sequence of actions by the user, a generative grammar[1] could be used to describe the command language.

The second and third requirements are a formalism for defining a set of compound data structure models, and a formalism for defining a class of line drawings which represent those models. Since we are describing things which are to be "constructed" (in a sense, generated) rather than analyzed, the use of something analogous to a generative grammar for defining classes of data structures and diagrams seems particularly appropriate.

The fourth requirement is a means of mapping from the command language definition to the model definition, and from the model definition to the picture definition. In other words, the user of the system would assume that use of a given sequence of statements in the command language would lead to the development of a particular model

1. See discussion in Appendix I.

and drawing; a different sequence of statements would lead to a different model and drawing. We need, therefore, a means of relating what will be said by the operator to the development of the required model and data structure.

The work discussed in this thesis deals with the third of these requirements, a method of describing classes of line drawings to the machine. In general, the problem becomes one of defining a set of machine primitives, specifying primitives for the given problem in terms of the machine primitives, and finally specifying the set of line drawings for the problem in terms of the specified problem primitives. For example, in the CALD system one problem primitive (which can be thought of as "machine primitives") is an AND-gate which is constructed initially from line segments. Use of the command language will result in the building of models and corresponding drawings based on these problem primitives. Obviously, an ad hoc system like CALD is built strictly for picture synthesis (or in a sense, "generation").

If a class of pictures can be described by something like a generative grammar in a system for picture recognition and analysis, then a formal system could be designed for picture synthesis using the same description. In following chapters, we shall consider a number of examples in which classes of pictures to be recognized have been defined in a formal way using generative grammars. Before

doing so, however, let us consider a simple example of a generative grammar (as used in linguistics) and the problems of extending this type of grammar to describe classes of pictures.

2.4 Generative Grammars:

It has been stated that some of the requirements of our system may be fulfilled by the use of a generative grammar. Therefore, we shall consider briefly an elementary example of a generative grammar as used by linguists in the study of the structure of languages.

The linguist determines a set of rules which may be used to generate the "basic" sentences of the language - this could be considered analogous to the building of only one model of the house. A typical, but very simple, grammar might have the following rules:

(1) sentence → NP VP

(2) NP → the N

(3) N → girl, boy

(4) VP → dances, writes, reads

The second statement, for example, is read "NP is replaced by 'the' followed by N". This grammar will generate a sentence like the one whose structure is shown by the tree in Figure 2.1.

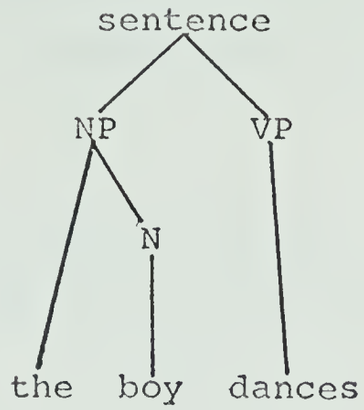


FIGURE 2.1

The linguist writes (or has as an objective the writing of) his grammar so that it will generate all the sentences in the language (or subset of a language) which he is studying. At the same time, it is also important that the grammar does not generate any sentences which are not well-formed sentences of that language.

It was stated that we are interested in generative grammars because we want to be able to define classes of pictures or, rather, classes of line drawings. Consider the following definitions:

(1) A LINGUISTIC GRAMMAR defines what is or is not a valid sentence of a language, and also provides a structured description of the sentence.

(2) A FORMAL GRAMMAR for programming languages is expressed as a set of productions which will generate all and only well-formed strings of the language. Again, the derivation of a string shows, as a syntax tree, its syntactic structure.

(3) A PICTURE GRAMMAR gives descriptions of a class of pictures by a set of productions used in a manner analogous to (2).

We could say that a picture grammar is an extension of the linguistic grammar from one-dimensional space to two-dimensional space. Of course, a picture grammar describes more complicated structures - in the spatial sense - than a linguistic grammar and therefore must have certain differences from the linguistic grammar. There seems to be two possible solutions to this difficulty: The first is to introduce more complex primitives and then use the simple operation of concatenation to join these primitives. The second is to introduce operations which are more complex than the simple concatenation operation. Both of these solutions are used in the picture grammars that will be discussed.

It will appear that a picture grammar, which is one form of a generative grammar, is a very good method for describing and generating classes of pictures.

CHAPTER 3 - PICTURE PROCESSING

3.1 Introduction:

A number of examples are available of the use of linguistic techniques to describe classes of pictures. This chapter discusses, as a first example, two articles written by R. S. Ledley (1962, 1964) which deal with certain aspects of picture processing. The term "picture processing" means, in this instance, picture recognition and the analysis of that picture; but the term could also include picture generation. The first article deals with the use of a computer to achieve artificial intelligence in abstraction which involves the problems of identification or recognition and of formulation. The second article discusses the pattern-recognition analysis of biomedical pictures.

3.2 Abstraction:

Ledley feels that the consideration of abstraction falls into two areas: The first is that of recognizing or identifying a concrete form as a member of a certain "abstract or conceptual class", a problem sometimes referred to as pattern recognition. For example, how does one recognize a certain animal as a "dog", or a combination of lines as an "A". The second consideration is that of "originally formulating the abstraction or concept from a plethora of experiences". (1962[1], p. 363) For example,

---

1. All references in the Chapter are taken from Ledley's published work.

how does one know, in the first place, that animals can be classified as "dogs", "cats", etc., or that there are groups of lines which form "letters".

The problem of identification may be solved by either deductive or inductive inference. "Deductive inference is the process of drawing consequences or conclusions from given premises, while inductive inference is the process of attempting to determine hypotheses or theories from which the given propositions may be deduced." (1962, p. 295) Ledley uses the term "deductive inference, as opposed to inductive inference, to describe intelligent behavior based upon deductive mathematical optimizing or regulating methods, as opposed to hypothesis making and intelligent trial and error." (1962, p. 324)

The object to be identified belongs to a certain class because of the configuration of its characteristics. Deductive inference using logical analysis could be a general method to discover if a picture is a member of a particular class. On the other hand, deductive inference could involve a syntactical-type analysis. For example, if we introduce certain syntactical relations, such as beside or inside, we would be able to construct the syntactical definitions of pictorial figures. If we define these relations then it would be possible to describe syntactically a simplified concept of a house. Table 3.1 shows the syntactical relations which are necessary for the definition of a house as shown in Table 3.2.

Table 3.1 SYNTACTICAL RELATIONS

Symbol	Meaning	Example	Representation
↑	is above	○ ↑ □	
→	is next to	○ → □	
⊙	is inside of	○ ⊙ □	
⊙	is inside of on the bottom	○ ⊙ □	
↥	rests on top of	△ ↥ □	
↦	rests next to	□ ↦ □	

Table 3.2 SYNTACTICAL DEFINITION

<door> → 

<windows> →  | <windows> →

<chimney> →  |

<wall> →  | <door> ⊙ | <windows> ⊙

<gable> →  | <chimney> ↥

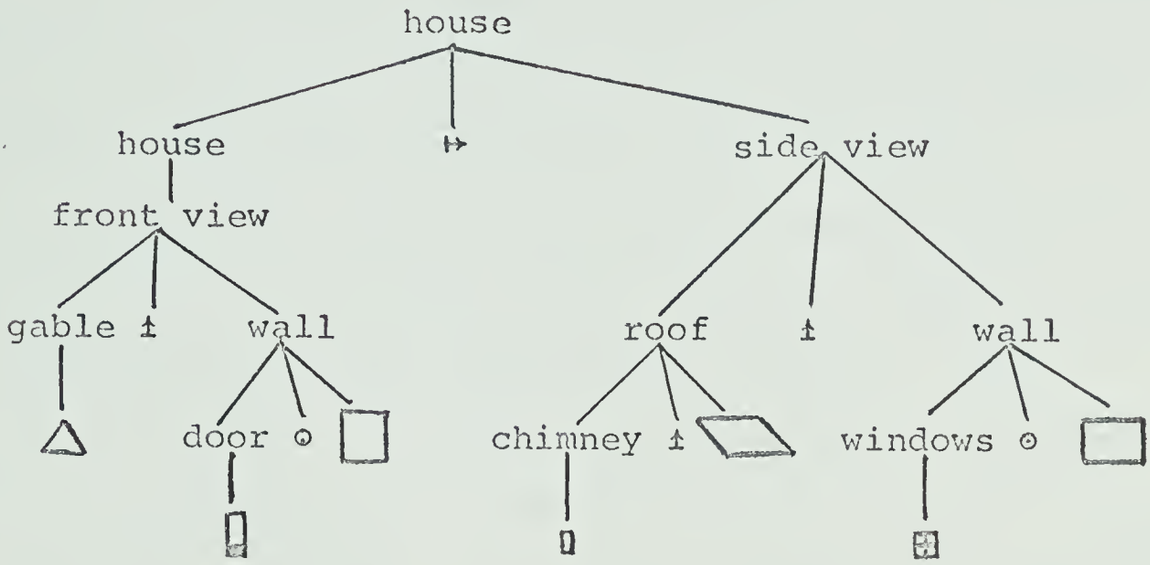
<roof> →  | <chimney> ↥

<front view> → <gable> ↥ <wall>

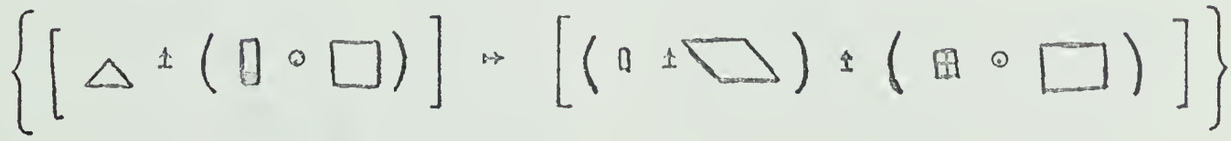
<side view> → <roof> ↥ <wall>

<house> → <front view> | <house> ↦ <side view>

If we were to apply these rules in the same manner in which they are applied in a linguistic grammar, we could build a tree structure like the one below.



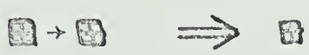
This tree structure gives us the following sequence of symbols,



which will generate a simple drawing of a house which looks like:

Therefore, in picture recognition using deductive inference, we start with the concept <house> and, by applying the syntactical definitions, we try to construct a figure which is the same as that which we want to recognize. Note that the syntactic operators (↥,→, ...) are analogous to (but more powerful than) the concatenation operator, which is implied (though not explicitly specified) within a linear string generated by a linguistic grammar.

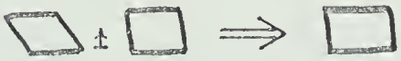
Inductive inference, in the form of heuristic programming, may also be used to identify an object. "Heuristic programming is the attempt to simulate on a computer the process of inductive inference" which is "the process of hypothesis making in a trial-and-error fashion, in order to draw conclusions or to solve problems." (1962, pp. 348-349) Using this method, the problem is to "eliminate the differences between the given object and the idealized concept of the abstraction" (1962, p. 365). Suppose we consider the houses as defined by the previous syntax. The problem then becomes to take the object string  and reduce it to  . We define certain operations as follows:

P1: 

P2: 

P3: 

P4: 

P5: 

P6: 

P7: 

where ⇒ means "is reduced to".

The procedure used to reduce the object string is relatively simple since it is cyclical in nature. The rules are scanned in order until a left-hand-side matches some part of the string. That part is then replaced by the right-hand-side of the rules. The rules are then scanned again, starting from the first rule. This process is complete (in success or failure) when none of the rules apply to the object string. If the rules were applied to

the above string, it could be reduced to the required string so that the process would be completed with a success.

The process of "pattern recognition" is closely analogous to the techniques used in syntax analysis of programming languages. The recognizer has a generative grammar definition of the set of pictures of houses (of the programming language), and attempts to recognize a given picture (source string). Though we are interested in the generation of a picture rather than the recognition of a picture, there is a logical connection which we shall discuss in section 3.4.

3.3 Description of Chromosomes:

One of Ledley's main interests is the analysis of biomedical pictures (1964), in this case the recognition of different types of chromosome. Certain abnormalities in chromosomes may indicate the presence of a disease; however the study of chromosomes by a trained technician, which involves making enlarged prints of chromosomes, cutting them out and aligning them with others so that they may be classified, is very time consuming and does not always reveal small differences. By using a computer to analyze the pictures, a chromosome can be recognized and analyzed in about half a second.

The picture to be analyzed is scanned by an optical reader and is encoded as a matrix of eight different

grey levels. The first phase of the analysis involves taking the matrix and scanning it using a "bug". If a point has a value less than a certain cutoff level, it is considered not to be part of an object; otherwise it is taken as part of the object. Using the cutoff level, the "bug" traces around the boundary of the object, always keeping the object to its right, and builds a list of characteristic parts describing the boundary. The characteristic parts are simply line segments for which curvature and direction are known. From this list of characteristic parts, the basic parts or primitives (Figure 3.1) of the object are determined. A syntactical description of chromosome shapes in terms of these primitives is defined by a generative grammar (see Table 3.3) and the chromosomes defined by this syntax are shown in Figure 3.2.

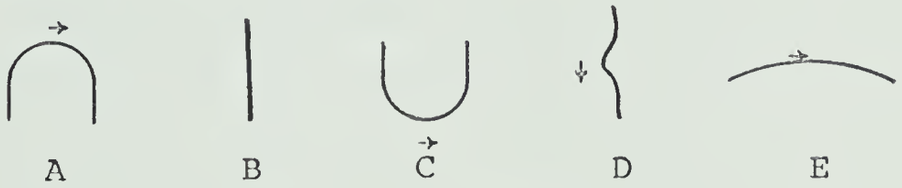


FIGURE 3.1 Primitives

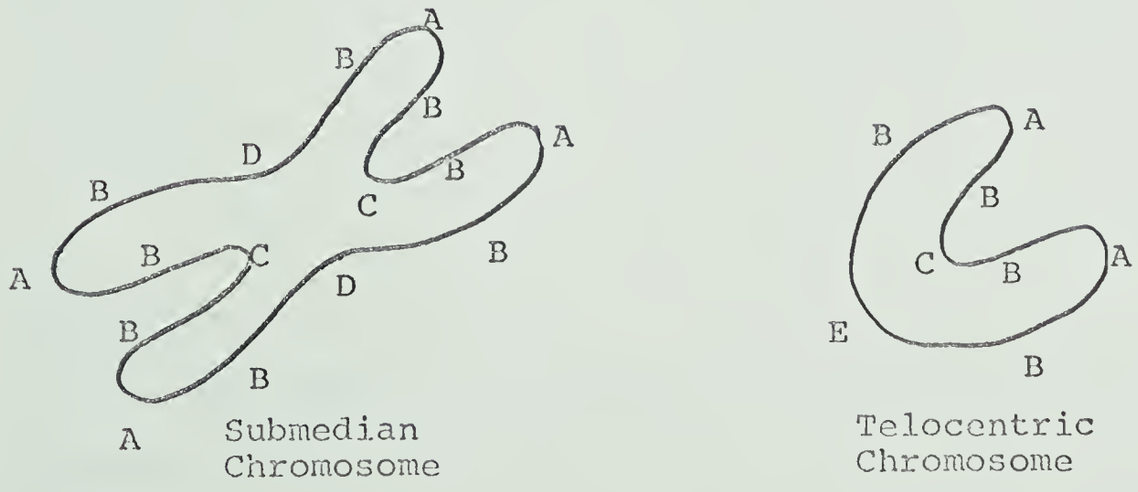


FIGURE 3.2

Table 3.3 SYNTAX FOR CHROMOSOMES

<arm> → B <arm> | <arm> B | A

<side> → B <side> | <side> B | B | D

<bottom> → B <bottom> | <bottom> B | E

<right part> → C <arm>

<left part> → <arm> C

<arm pair> → <side> <arm pair> | <arm pair> <side> |
<arm> <right part> | <left part> <arm>

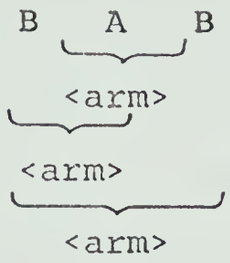
<submedian chromosome> → <arm pair> <arm pair>

<telocentric chromosome> → <bottom> <arm pair>

In the second phase, the generative grammar is used by a "bottom-up" syntax analyzer which either identifies the list as describing some type of chromosome, or finds that the list does not describe a chromosome. In the latter case, the bug will go on to the next object. If the list does represent a chromosome, then a derived list is processed to find, for example, the length or the area of the chromosome.

The analyzer uses the list of basic parts of an object, applies the grammar to this list and attempts to reduce the list to a chromosome. For example, consider the sequence B A B. From the syntax, A is an <arm> so we have B <arm> B. Now B <arm> is an <arm>and <arm> B is an <arm>

so that B A B is an <arm>.



This is an example of the heuristic technique discussed in the previous section.

3.4 Discussion:

The important aspect of Ledley's work from our point of view is the way in which he has adapted linguistic techniques to the definition of two-dimensional pictures.

3.4.1 Syntactical Definition of Houses:

Ledley has introduced two concepts to solve the problem of adapting linguistic grammars to describe classes of pictures. First, the primitives are more complex than those of a linguistic grammar in that they are two-dimensional. This implies that a recognizer or a generator is able to handle objects which are not linear. Second, certain syntactical relations are introduced to extend the linear operation of concatenation. These relations allow objects to be placed in relative positions other than one after the other.

The syntactic definitions and syntactic relations seem to be adequate for picture recognition but are not complete enough for generation. The main problem arises

from the fact that we cannot place any conditions on the syntactical relations. For example, the relation "is inside of" does not specify that the first object needs to be smaller than the second object. This is obvious to the human user but must be specified in detail for use in a computer system. Other problems arise from the fact that there is no way to specify the size of the primitive or the distances between primitives in relations such as "is above". The distance problem may be easily solved by saying that the distance can vary from a fraction of an inch to the maximum allowed by the size of the display area. There are two possible solutions to the problem of size: (1) the primitives are a given size or (2) the primitives may be any size limited only by the dimensions of the display area. The first solution seems unnecessarily restrictive while the second solution could allow, for example, the roof to be much larger than the wall.

It is obvious that although this grammar may be adequate as far as a human user is concerned, it is not powerful enough for use to describe adequately to a machine a wide range of line drawings. It is possible that certain elements could be added to the grammar, say "attributes" as defined by Narasimhan (see Section 4.1).

3.4.2 Grammar for Chromosomes:

A generative grammar is used to define the class of pictures which the system must recognize and analyze.

The problem of adapting a generative grammar to define two-dimensional objects is solved by defining the primitives as two-dimensional objects, and by defining only the outline of these quite simple shapes. The pictures may then be characterized by a linear string of these primitives.

This system recognizes only well-formed chromosomes and any other shapes are simply ignored. The size of the objects should not present a problem since the primitives are defined in terms of curvature as well as length. Also the objects to be recognized are of relatively the same size. The problem of specifying distance that was seen in the "houses" grammar does not arise here since the primitives are simply joined end-to-end and there are no distance relations involved. It appears that this class of pictures is well-defined and that the method of definition is rigorous enough to be exploited on a computer.

This system recognizes a chromosome by its boundary characteristics and the assumption is made that the interior of the object is solid. Since the line segments used to define the boundary are easily adapted to the vector drawing of a graphical display, we need simply redefine the primitives to be thin line-like elements so that we can use this grammar for a graphical system. Of course, there will be problems when it is desirable to overlap two objects but that is minor compared to the problems which would arise in trying to adapt a grammar like

the one for generating houses. This method of definition, though, is restrictive in that it does not allow us to specify primitives joined other than end-to-end.

The question as to whether Ledley's method gives us any insight into the problem of developing more useful software will be discussed in later chapters.

CHAPTER 4 - SYNTAX-DIRECTED INTERPRETATION

4.1 Introduction:

The second example of the use of linguistic techniques to describe classes of pictures is from the work of R. Narasimhan (1966). Narasimhan was interested in the description of a picture when it was not possible to classify the picture according to a prototype or model of the class of pictures to which it belonged. His research has involved "... investigating the possibilities of syntax-directed interpretation of classes of pictures" (1966, p. 167), where the classes of pictures were limited to those containing thin line-like elements.

Narasimhan felt that pattern recognition was only part of the problem involved, since recognition only implied a decision that the picture was or was not a member of the class of pictures being studied. A more important objective was to be able to give a description of the picture which would resemble that given by a man after manual study of the picture. He also felt that any processor which could produce, for the input picture, a structural description from which verbal descriptions could be mapped, must use a generative grammar.

The process which Narasimhan developed to produce a structural description of a picture was based on a system for hierarchical description. Any particular system has a

given set of primitives, each primitive having a list of attributes. An example of an attribute would be the coordinates of a point. The class of pictures is described in a sequence of steps where an object type is described in terms of primitives, more complex object types are described in terms of these simple object types, and finally the picture is described in terms of complex object types.

We shall consider two of Narasimhan's examples: The first system was designed for the purpose of describing bubble chamber pictures. The second system was built to generate a subset of the English alphabet.

4.2 Bubble Chamber Pictures:

A bubble chamber picture is a photograph of the particle track configurations from a physics experiment. The picture is digitized into a square array of points, with each point either being in a track or not in a track. The purpose of the system is not simply to recognize objects in these pictures, but to give a structural description of the picture (and from this a verbal description) so that a physicist would be able to choose the pictures which were of interest to him. An example of a small bubble chamber picture which has been preprocessed to fill in gaps and to eliminate isolated points is shown in Figure 4.1.

The set of primitives, which we will consider to be the first level of the hierarchy, consists of one object type - the POINT. The attribute list of any point contains the grey-level (in this case, black or white) and the coordinates of that point. At the second level of the hierarchy, black points which make up straight line segments are named ROAD SEGMENTS. The attribute list of a road segment consists of the directional orientation which may be north-south, east-west, right-diagonal, or left-diagonal with the labels N, E, A, and B, respectively.

At the next level, the nodes where the road segments intersect or meet are defined as MULTI. The end points of a road segment are labeled as TERMINALS. A multi may then be further categorized into BEND, CROSSOVER and VERTEX. "The junctions, crossings, bends, etc. where two or more roads meet, are identified by their multiple labels. In the outputs these are given by the following code:

EA:	3	AB:	0	ENB:	4
EN:	5	NB:	2	ANB:	8
EB:	9	EAN:	7	EANB:	U
AN:	6	EAB:	1		

The points which have not been assigned any of the labels are indicated by asterisks (*)." (1964, pp. 165-166) An example of the labeled output is shown in Figure 4.2.

Finally, at the highest level, we have an EVENT of a particular picture which has certain attributes. Using this hierarchy of labels, which is a structural description,

it is possible to describe a bubble chamber picture according to the attributes which the user feels are of the most interest to him.

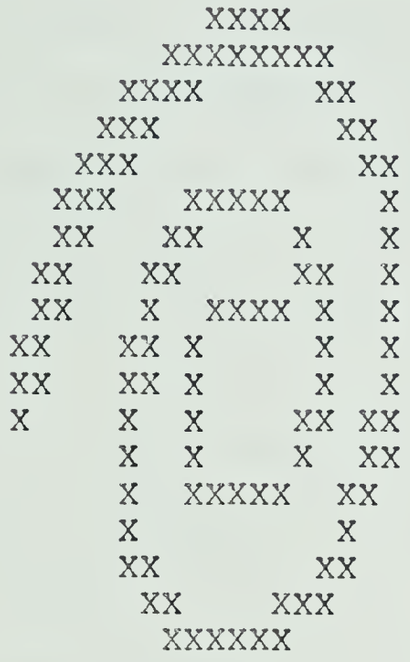


FIGURE 4.1

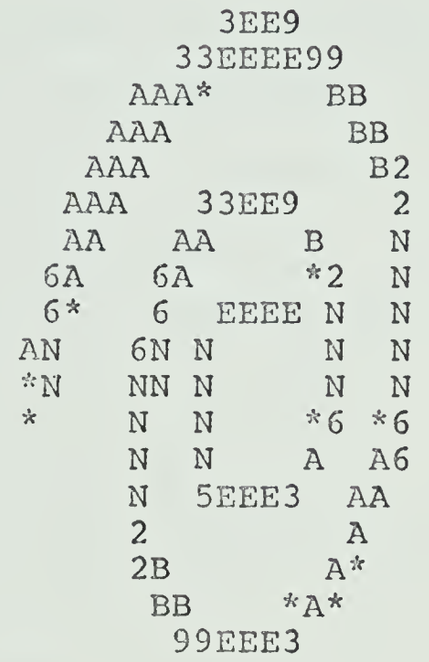


FIGURE 4.2

4.3 Generation of English Letters:

A system for generating a subset of the English letters was written by Narasimhan to allow the computer to draw English letters, so that, for example, the computer could be used in designing posters. The hierarchical structure is not as evident as that of the system for describing bubble chamber pictures but, nevertheless, does exist. On the other hand; the process, used to build the structural description of a letter, quite obviously uses a generative grammar.

The primitives (or words) of this system are the line segments as shown in Figure 4.3. Each primitive has a pair of attributes assigned to it: length (tall, medium, or short) and thickness (thick, medium, or thin). Also, each of the elements or phrases of the grammar has a distinct set of vertices (which may be empty) corresponding to it. "The vertices are the points where a word or phrase can be linked to another word or phrase to form a phrase. The vertex set of the resulting phrases is a specific subset of the union of the vertex sets of its component phrases." (1967, p. 281). Narasimhan has used the terms "word" and "phrase" instead of the more usual terms "machine primitive" and "problem primitive". He chose these terms in order to show the analogies between his system and linguistic grammars.

A partial set of the rewriting rules of the generative grammar is given in Table 4.1. In order to illustrate this system, let us consider the generation of the letter A. First the primitives r and l are joined to form the object type INVE and then the primitive h is joined to INVE to form the letter A. (See Figure 4.4). Obviously, there are two levels to the hierarchy. The construction of some letters requires more levels while for other letters, like C, only one level is involved.

Table 4.1 REWRITING RULES

Leter → A|B|C|D|E|F|H|I|L|N|O|P|Q|R|S|T|V|X|Z

A → INVE.h(11,23;) | INVH.h(11,23;)

INVE(1,2) → r.l(11;2;2) | r.v(11;2;2) | v.l(11;2;2)

INVH(1,2) → SGMMA.l(31;1;2)

SGMMA(1,2,3) → r.h'(11;2,3;3)

B → PE.d'(11,23;)

PE(1,2,3) → v.d'(11,23;2,3;2) | r.d'(11,23;2,3;2)

C → c

D → v.d(11,33;) | r.d(11,33;)

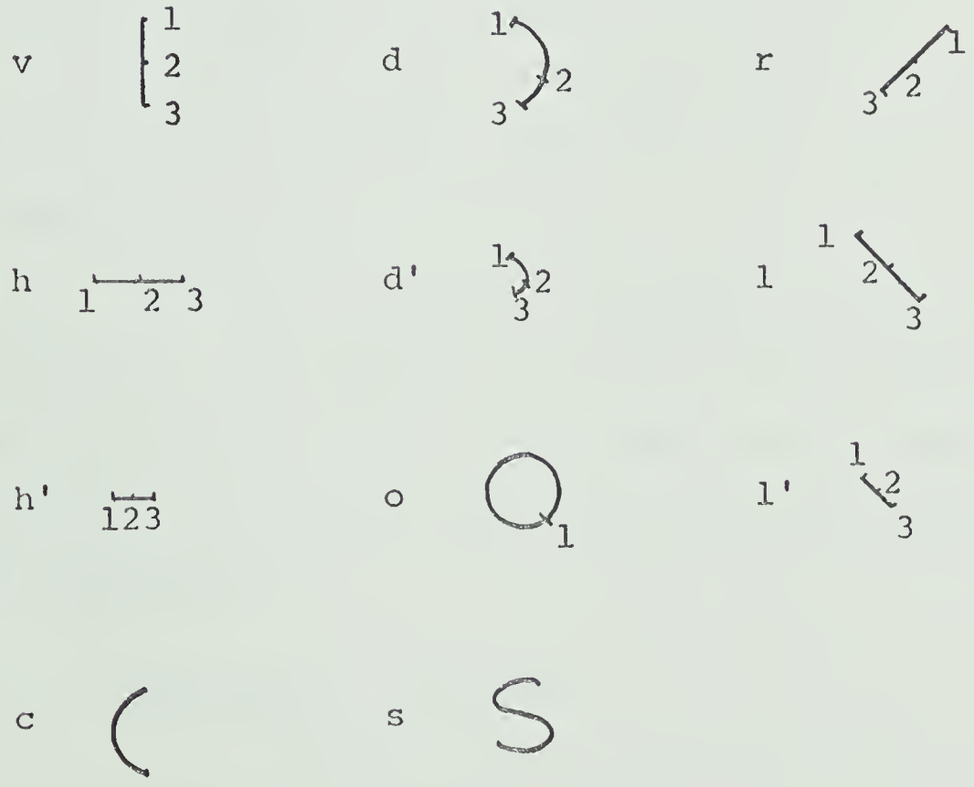


FIGURE 4.3

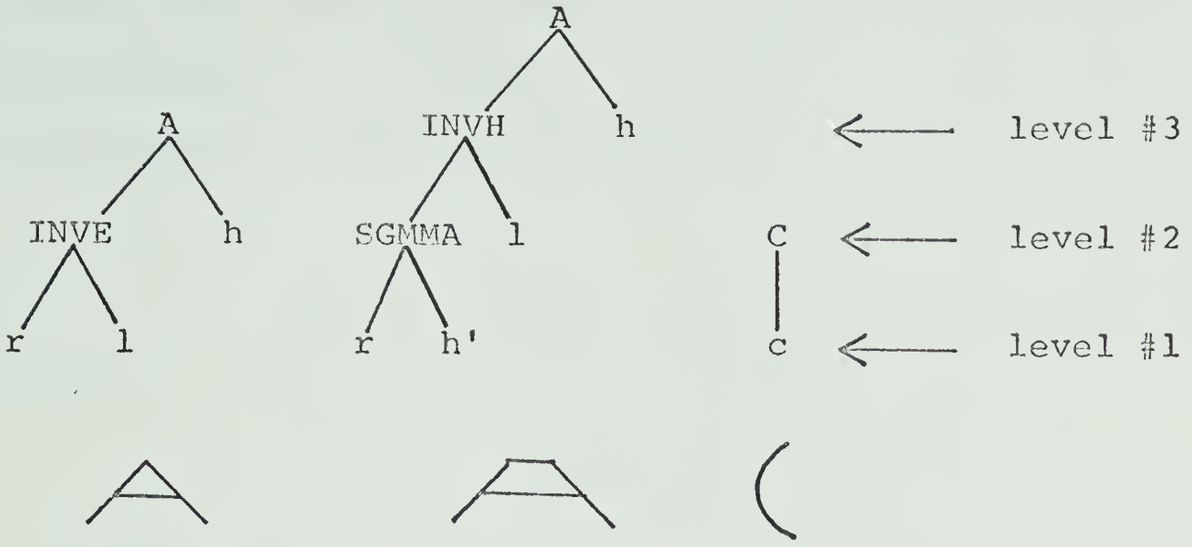


FIGURE 4.4

The rewriting rules are set up in the following manner:

(1) The numbers in brackets on the left-hand side indicate the number of vertices in the result.

(2) The first numbers (actually pairs of digits) in brackets on the right-hand side tell which vertices of the two symbols specified are joined.

(3) The next n numbers correspond to the n vertices of the result. The numbers after the first semi-colon, but before the second, refer to the first symbol specified; the numbers after the second semi-colon refer to the second symbol specified.

For example, consider

SGMMA(1,2,3) → r.h'(11;2,3;3).

The result has three vertices and is constructed by joining the first vertex of r to the first vertex of h'. In the

result, the first and second vertices correspond to the second and third vertices of r, while the third vertex corresponds to the third vertex of h'. (See Figure 4.5)

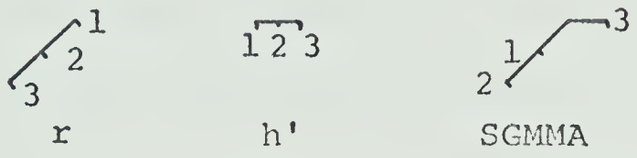


FIGURE 4.5

4.4 Discussion:

The interesting aspects of Narasimhan's work, as far as we are concerned, are the hierarchical structuring and the introduction of the attribute lists. Also of interest are the ways in which he has overcome the problems of describing two-dimensional objects.

4.4.1 Bubble Chamber Pictures:

It was stated earlier (in Chapter 2) that graphics software allows the programmer to specify how classes of pictures at one level may be constructed from those at lower levels. That obviously involves a hierarchical system. Narasimhan's system may offer ideas on improving existing software and we shall consider this particular question later in Chapter 7.

The attribute list allows certain information to be specified about an object type at any level. This is important for two reasons: First, an object type may be

specified in considerable detail. Second, the hierarchical structure is easier to define since the attribute list of an object type at a given level is a function of the attribute lists of objects at lower levels. Also, the more information that can be specified as functions of variables, the less information the user will have to supply to the system.

The problem of describing two-dimensional objects was solved by using one-dimensional primitives and introducing spatial relationships. Hence, we have one-dimensional primitives being joined to form two-dimensional object types in higher levels of the hierarchy.

4.4.2 English Letters:

In the generation of English letters, the problem of describing pictures has been solved by the introduction of two-dimensional primitives. Also, the joining or connecting operations are not simply the linear operation of concatenation, but allow connections at various points of a primitive. One is therefore allowed to specify what points of two objects are joined and where the vertices of the new object will be placed. A possible extension of this would be to allow one to specify other variables, such as size.

It seems that a system like this has certain advantages over the one for describing chromosomes, since it allows objects to be joined at points other than the end points of the primitives.

CHAPTER 5 - PICTURE DESCRIPTION LANGUAGE

5.1 Introduction:

This chapter deals with a Picture Calculus developed by W. F. Miller and A. C. Shaw (1967,1968). The Picture Calculus is a formalism for both recognizing and generating pictures. "It is comprised of a picture description language (PDL), rules for transforming and comparing structures represented in PDL, data structures and control for generation of pictures, and the parsing and primitive recognizers needed for picture recognition." (1968, p.102).

The picture calculus has been developed to apply to a restricted class of pictures, namely "... those pictures that can be described by graphs, where edges represent picture components and the graph connectivity mirrors component connectivity." (1967, p. 4). Some of the types of pictures being studied are pictures from high energy physics experiments (i.e., bubble chamber pictures), characters from various alphabets, line drawings such as flowcharts, and biomedical pictures.

The most interesting part of the picture calculus, and the only part which we shall discuss, is the picture description language. The PDL is a language for writing picture descriptions, that is, a language for writing productions of picture grammars. The PDL has a syntactic

definition, given by what might be considered a grammar of a language for writing picture grammars. According to Shaw (1967, p. 3) "... a good picture description language should provide a model within which a large class of pattern recognition problems can be solved." He feels that such a formalism is to be used:

(1) in the description of pictures (and discussion about pictures) by humans.

(2) as the basis of a computer system for recognition and analysis.

(3) for the computer generation of pictures.

We shall look at the picture description language in some detail and consider briefly an example of a class of pictures which is described using the PDL.

5.2 Picture Description Language:

The two basic components of the picture description language are a means of describing classes of primitives and a set of operators for joining and transforming the primitives.

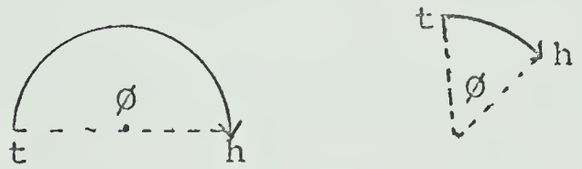
5.2.1 Primitives:

A primitive is a two-dimensional (potentially an n-dimensional) object with two specified points, a head and a tail. Usually, a primitive is an object which may be more easily recognized (and generated) as a unit than as a

list of its parts. Primitive pictures may be concatenated only at the heads and/or the tails.

Any primitive picture is a member of a primitive class specified by an attribute list. The attribute list for a primitive class is made up of a class name, specifications for the head and the tail, and a list (possibly empty) of additional attributes. If P is a primitive class, then P' is the primitive defined by a certain specified attribute list of the class P.

As an example, we may define the primitive class of arcs of circles with any radius, negative curvature, and angle less than 180°.



The attribute list of this primitive class has the form:

ARC → (ARC, counterclockwise limit, clockwise limit, curvature < 0, $\emptyset < 180^{\circ}$)

This example is a particular instance of the general form of the attribute list:

PRIMITIVE CLASS → (<NAME>,<tail specification>,<head specification>,<1st attribute>,..., <nth attribute>)

The value list for a specific element ARC' of the primitive class ARC would be given as:

VALUE(ARC') → (ARC', $\vec{x}_{tail}$, $\vec{x}_{head}$, (curvature=) -2, ($\emptyset$=) 60°)

There are certain special primitives: the blank primitive, the don't care primitive and the null point primitive, λ. The null point primitive consists of a head and a tail at the same point.

5.2.2 Syntax of the PDL (1968, p. 104):

"A Sentence, S, in the language is defined by

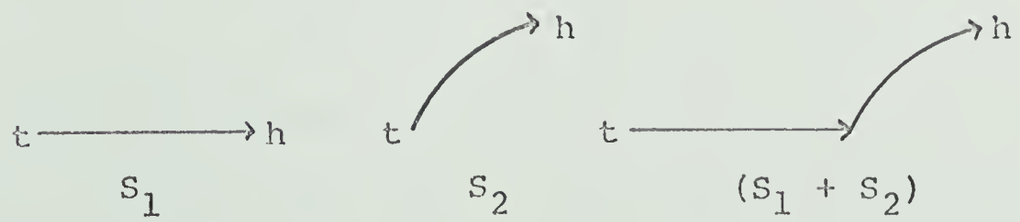
1. $S \rightarrow p \mid (S\ \Theta\ S) \mid (\sim S) \mid (\overline{S}) \mid T(\omega)S \mid S^{l}$
2. $\Theta \rightarrow + \mid x \mid - \mid * \mid \sim$
3. p is a primitive class
4. [+,x,-,*] are concatenation operators
5. [~,^{r},T(ω)] are unary operators
6. l is a label designator."

5.2.3 Concatenation Operators:

The concatenation operators described below are binary operators and in all cases $TAIL((S_1\ \Theta\ S_2)) = TAIL(S_1)$ and $HEAD((S_1\ \Theta\ S_2)) = HEAD(S_2)$ where $\Theta \in [+,x,-,*,\sim]$.

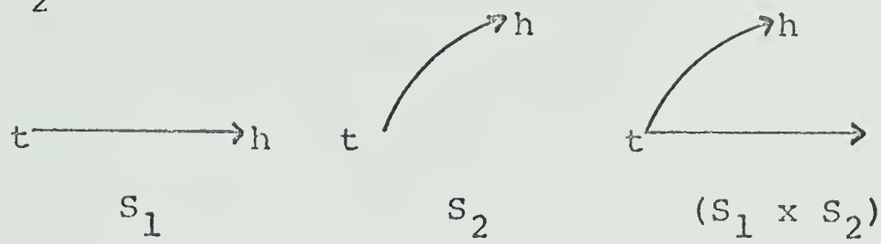
(1) The + operator:

$(S_1 + S_2)$ concatenates the head of S_1 with the tail of S_2.



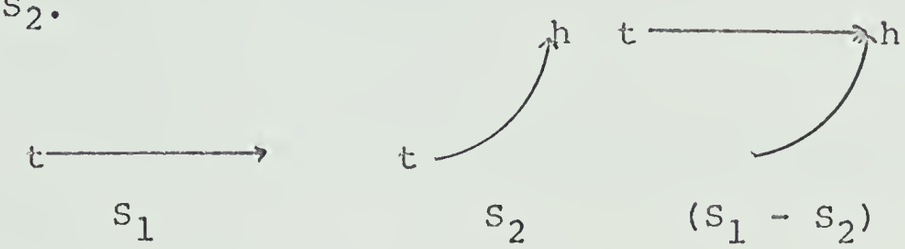
(2) The x operator:

$(S_1 \times S_2)$ concatenates the tail of S_1 with the tail of S_2.



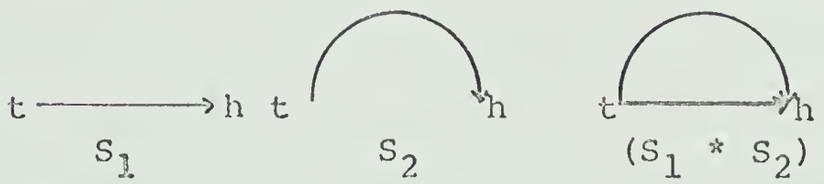
(3) The - operator:

$(S_1 - S_2)$ concatenates the head of S_1 with the head of S_2.

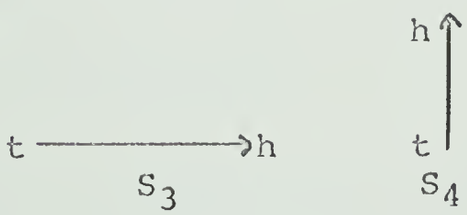


(4) The * operator:

$(S_1 * S_2)$ concatenates the tail of S_1 with the tail of S_2 and the head of S_1 with the head of S_2.



Obviously the operator * will be undefined for some primitives. For example, $S_3 * S_4$ will have no meaning for S_3 and S_4 defined as:



(5) The binary ~ operator:

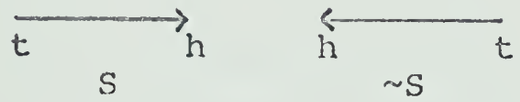
$(S_1 \sim S_2) = (S_1 - (\sim S_2))$ where $(\sim S_2)$ is defined in secton 5.2.4.

5.2.4 The Unary Operators:

(1) The unary ~ operator:

HEAD((~S)) = TAIL(S)

TAIL((~S)) = HEAD(S)



(2) The ⌜‾ operator:

HEAD((⌜S)) = HEAD(S)

TAIL((⌜S)) = TAIL(S)

All points in the structure are changed to null points.

(3) The Transformation T(ω) Operator:

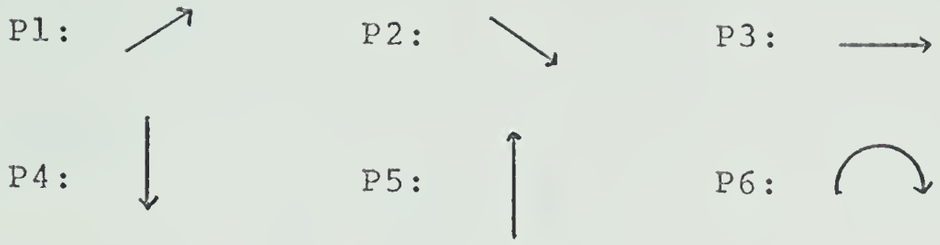
$$" \vec{X}' = M\vec{X} - \vec{T}$$

where $\vec{X}$ is any point in the structure, $\vec{X}'$ is the corresponding point in the transformed structure, M is a matrix of constants, and $\vec{T}$ is a constant vector. This includes stretching, rigid body rotations and translations, and shearing transformations." (1968, p. 107).

5.2.5 Illustration:

As an illustration of the way in which the PDL may be used, consider the following description of two types of houses - one with a round roof and one with a triangular roof - as they might be described by a young child. The primitive classes, each with a single primitive,

are:



The two types of houses are described by the following sentences in the picture description language:

H → (R * B)

R → R1 | R2

R1 → ((P1 + P2) * P3)

R2 → ((P6 * P3))

B → ((P4 + P3) + P5)

For example, the string of concatenated primitives ((P6 * P3) * ((P4 + P3) + P5)) describes a house with a round roof.

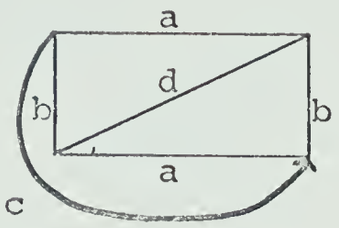
5.2.6 Labeling:

In the above illustration, each primitive belonged to a given class and these primitives could be differentiated. However, for two primitives from the same class, there is no method for choosing one before the other. In some cases, as in the example below, this differentiation is necessary, so the concept of labelling has been introduced. "The labelling scheme is edge-oriented and preserves the identity of the edge through the various transformations and

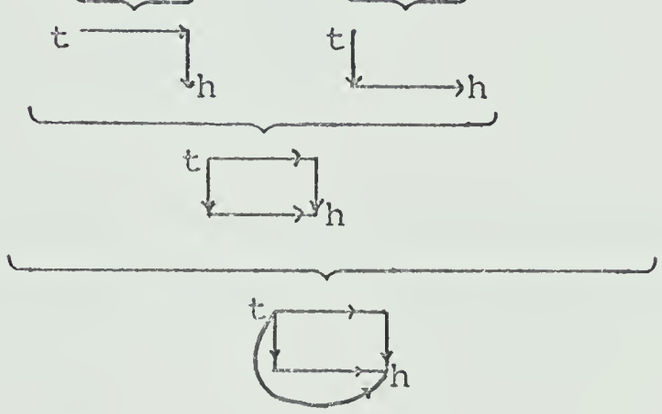
manipulations which operators and operands may undergo." (1968, p. 111)

EXAMPLE:

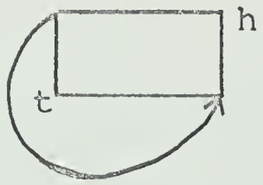
P":



P → (((a + b) * (b + a)) * c)



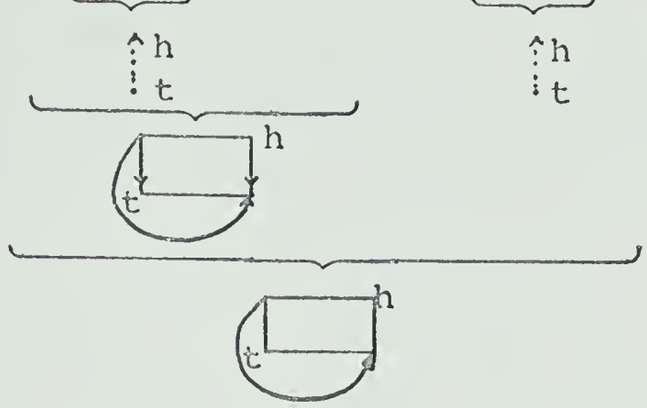
P':



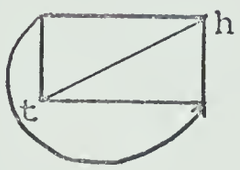
The problem is to transform P into P'. The solution is to label the first b with β and to label the second b with α so that P → (((a + b^{β}) * (b^{α} + a)) * c). Now we will require three statements to construct P".

P → (((a + b^{β}) * (b^{α} + a) * c)

$$P' \rightarrow (((\overbrace{(\sim b^{\alpha})}^{h\ t}) + P) + (\overbrace{(\sim b^{\beta})}^{h\ t}))$$



$$P'' \rightarrow (P' * d)$$



Therefore, it is possible to change the head and tail positions of a picture so that primitives may be added where they could not have been joined before.

5.3 Discussion:

Miller and Shaw have introduced a formalism which may be used to construct a generative grammar for describing a certain class of pictures. The description of a picture is essentially a linear string of elements (i.e., primitive classes and operators) which can be analyzed by the already existing methods used for string languages. The problem of describing two-dimensional pictures has been solved by a combination of methods similar to those used by Narasimhan in the generation of English letters. The primitives are defined to be two-dimensional objects and the operators are more complicated than simple end-to-end concatenation.

The idea of using primitive classes, rather than primitives, seems very practical. In our other examples, we had the problem of specifying different sizes and of making sure that the sizes of two primitives were compatible. In this grammar, it is possible to define a primitive class; then, each time a primitive is used, its size can be specified. For example, suppose we define the primitive class of straight lines with angle of 45°. Each time this class is used, the head and tail (and therefore the length) are specified. This still leaves us the problem of compatibility. However, an attribute of size could be added to the list description, and the sizes of any two primitives could then be compared for compatibility. An attribute of size for a primitive could be expressed as a function of head and tail coordinates. However, if a primitive is to be concatenated to a sentence, the size attribute of the sentence based on head and tail coordinates may not be accurate. Therefore, the size attribute should be stated explicitly and not as a function of head and tail coordinates.

Although the concept of classes of primitives seems useful, it is possible that having only two vertices on any primitive or any concatenated picture may present problems. For example, consider the CALD system. One of the primitives of this system is the AND-gate (see Figure 5.1). Other primitives may be joined to an AND-gate at vertices A, B or C. Since the PDL only allows two vertices on any

primitive, we would need to find some way for describing an AND-gate with only two vertices instead of three. Obviously, it is sometimes inconvenient to have only two points of concatenation on a given picture.

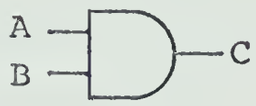


FIGURE 5.1

This formalism has excellent possibilities, since Miller and Shaw have shown that it can be used for specifying construction rules of houses, to define chromosomes and to define English letters. However, the particularly important concept for our work is the introduction of classes of primitives.

CHAPTER 6 - PICTURE DESCRIPTION IN GRAPHICS SOFTWARE

In preceding chapters, a number of formal methods for defining classes of pictures have been discussed. Within any graphics software, there are facilities for defining classes of pictures, although these facilities are not likely to have any formal basis for their design.

The description of a class of pictures in graphics software can be divided into two distinct parts: The construction of the problem primitives, and the definition of the class of pictures based on the primitives. We have already considered briefly the hierarchical structure of the design problem (see section 2.2) and we shall take a closer look at it now.

The hierarchy consists of four levels with each level being dependent upon the other levels (Figure 6.1) of the hierarchy. For example, the location of a machine primitive depends on the location of the problem primitive which in turn depends on the location of the picture. The CALD system illustrates the hierarchical structure of pictures that is found in design systems. The machine primitives are point, vector and symbols: the problem primitives, defined in terms of the machine primitives, include NOT-gate, AND-gate and OR-gate. These primitives may be joined in various ways to form subcircuits, for example, a half-adder; and the subcircuits joined to form

complete circuits, for example a (full) adder. In this chapter, it is intended to discuss only the description of problem primitives in terms of machine primitives.

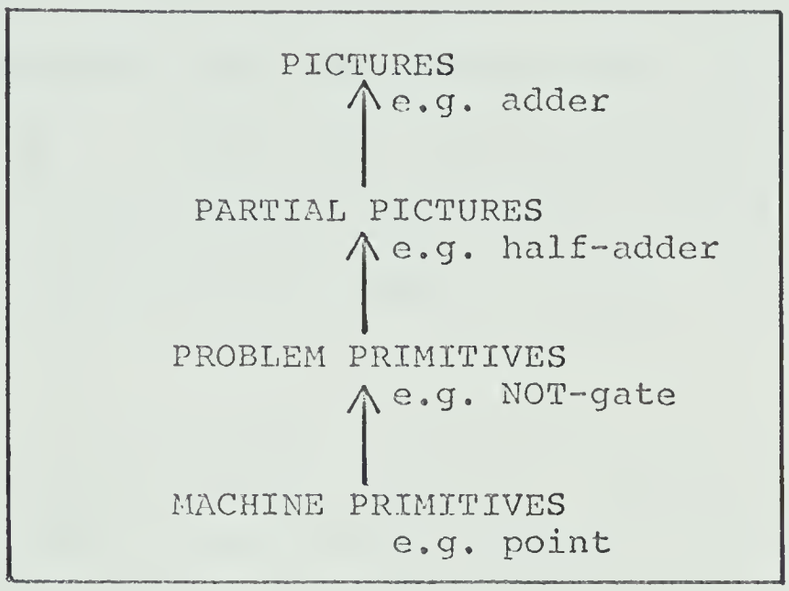


FIGURE 6.1

In some cases, the definition of problem primitives is very explicit. For example, the definition of a logic circuit primitive, the NOT-gate, in CALD has the form:

```
CALL BLOCK (1)
CALL VECTOR (A, 15, 0, NB, ND)
CALL VECTOR (A, 15, 10, NB, ND)
CALL VECTOR (A, 35, 0, NB, ND)
CALL VECTOR (A, 50, 0, NB, ND)
CALL MOVE (A, 35, 0)
CALL VECTOR (A, 15, -10, NB, ND)
CALL VECTOR (A, 15, 0, NB, ND)
CALL ENDBLK
```

The shape, size and orientation are specified in each block while the display position is allowed to vary. Because a primitive can be displayed anywhere, each block definition actually defines a class of pictures (which are relatively trivial by comparison with the overall class of pictures being described). In this same logic circuit design system, the definition of the class of allowable circuit diagrams is in no way explicit but is implicit in the structure of the system.

We can now ask the question: Is there any insight to be gained from the more formal methods of defining classes of pictures (discussed in Chapters 3, 4 and 5) which might suggest ways of designing better software for a graphics system?

In Chapter 7, a simple shape will be defined using each of the formal methods discussed and then an attempt will be made to answer the above question. First, however, a simple shape (the same as will be used in the next chapter) will be defined using the facilities of GRIDSUB (discussed in Section 2.2). The picture to be described is the class of rectangles of any dimensions where the size is limited only by the size of the display area. For simplicity, we shall consider rectangles which are parallel to the axes of the display area.

The first step in defining an object (or, as we have pointed out, a limited class of objects) in GRIDSUB is to construct a BLOCK which includes the lines, points and

symbols necessary to form that object. The definition of a rectangle requires four straight lines which are joined end-to-end at right angles and form a polygon. The following set of GRIDSUB CALL's will describe a rectangle, given certain conditions which will be described later.

```
CALL BLOCK (1)
CALL MOVE (F,X1,Y1)
CALL VECTOR (F, X2, Y2, F, F)
CALL VECTOR (F, X3, Y3, F, F)
CALL VECTOR (F, X4, Y4, F, F)
CALL VECTOR (F, X1, Y1, F, F)
CALL ENDBLK
```

These instructions define the class of rectangles.[1]

The MOVE routine defines the movement of the beam, without a line being drawn, to the position X_1,Y_1 on the screen where X_1 and Y_1 are relative to the origin of the block (which will be defined at display time).

The VECTOR routine defines a straight line from the present position of the beam to the specified X,Y position. These points are specified with respect to the origin of the block. In order for the lines constructed

---

1. The instructions BLOCK to ENDBLK inclusive define a graphic entity.

by the vector routines to form a rectangle, the values of X_i and Y_i must conform to certain conditions. The conditions are:

(1) $X_1 = X_2$, $X_3 = X_4$, $Y_2 = Y_3$, and $Y_4 = Y_1$

OR $X_2 = X_3$, $X_4 = X_1$, $Y_1 = Y_2$, and $Y_3 = Y_4$

(2) $0 \leq X_i, Y_i \leq 1023$ - this is required since there are only 1024 possible positions on the X and Y axes of the graphical display.

Once all the values have been specified, the structure defined by the block can be displayed using the following instructions:

CALL DISPLY (1, N, K, L)[1]

CALL TRANMT[2]

The display routine builds that display file for BLOCK 1, with identification number N and with origin at (K,L) of the graphical display. The TRANMT routine then sends the display file to the graphical machine where it will be displayed.

---

1. A set of these is used to assemble a complete picture which is to be shown on the graphical display.

2. This call specifies the end of the picture assembly and causes the picture to be shown on the CRT.

The routines of GRIDSUB allow the explicit definition of a class of objects which are constructed from four straight lines. However, the restrictions imposed on the four lines (in order to construct a rectangle) are not explicitly stated but must be incorporated in the program. That is, the programmer will code the program so that only a rectangle may be generated from the four straight lines.

The fact that the definition of an object is implicit in the program rather than explicit is typical of most graphics systems in use today. They were built to solve the description problem without any thought to a formal basis, and are dependent upon the hardware which is used. Hence, the systems are ad hoc in their nature and, in most cases, difficult to modify or extend.

Hopefully, the description of rectangles (in the next chapter) using more formal methods will offer some insight into the problem of developing more satisfactory methods of picture description within practical graphics software.

CHAPTER 7 - DEFINING A SIMPLE SHAPE

7.1 Introduction:

The question has been asked: Can formal methods be used to improve graphics software? In order to answer this question, we will compare the definition of a simple shape (the class of all rectangles) in the existing graphical software with the definitions of that same shape using the more formal methods already discussed.

7.2 Definition Using Ledley's Method for Chromosome Description:

The shape of a chromosome is described as a string of primitives with the end of one primitive in the sequence joined to the beginning of the next primitive. In order to describe a rectangle as a sequence of primitives, it is necessary to introduce a set of suitable primitives. These are not necessarily machine primitives, as described in Chapter 6, but generally larger components definable in terms of machine primitives, and considered more useful for constructing problem primitives. The primitives required are a "line segment" (Figure 7.1a) and a "right angle" (Figure 7.1b) where each primitive has a number of possible orientations. The line segment has two orienations (0 and 1) and the right angle has four orientations (0,1,2 and 3) (Figure 7.1c).

If we were to define these using GRIDSUB, we would need to consider the fact that the machine primitives of the graphical display are points, lines and symbols. Therefore, it is necessary to define each of these primitives as a block (or something like a block). Each block definition will allow the user to specify location but it is not necessary to allow the user to change the size or the orientation of these primitives.

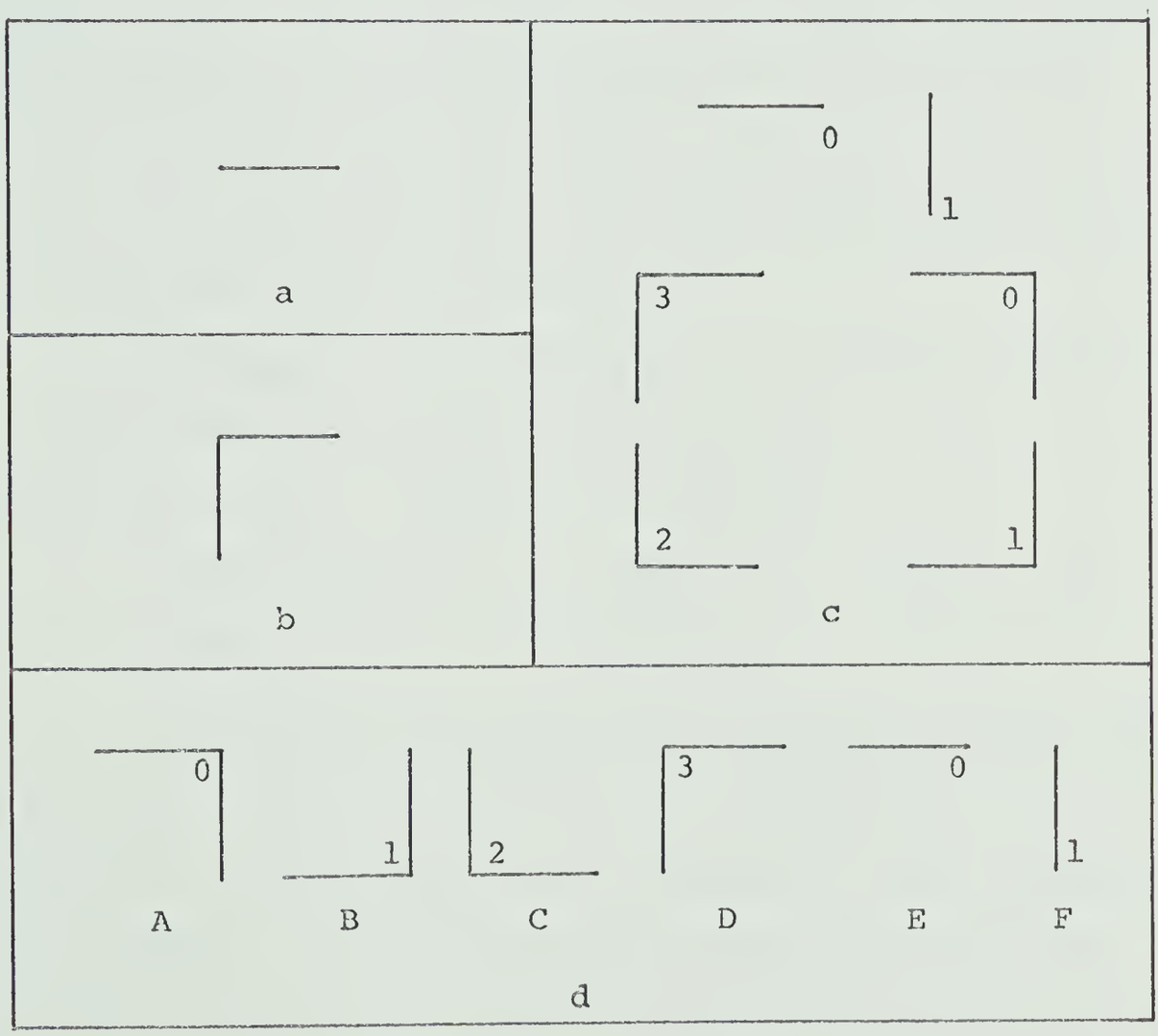


FIGURE 7.1

Ledley has not introduced any notation for allowing different orientations of a primitive nor for numbering primitives. (Ledley did not need these features since in his problem of recognition of chromosomes the orientations of the objects could vary without affecting the interpretation. In the case of graphics, the problem is to synthesize things which must be precisely definable without such variations.) Therefore, each of the six figures in Figure 7.1c will be considered a primitive and named A, B, C, D, E and F (Figure 7.1d). For simplicity, it is assumed that description and generation of a rectangle starts at the first line segment, E, after the right angle, D (Figure 7.2). A grammar which will generate the class of rectangles might look something like the following:

<rectangle> → <top triangle> B <bottom triangle> D

<top triangle> → <side 1> A <side 2> | A

<bottom triangle> → <side 1> C <side 2> | C

<side 1> → E | E <side 1> | <side 1> E

<side 2> → F | F <side 2> | <side 2> F

A problem arises in that the opposite sides must be the same length in order to form a rectangle (i.e., <side 1> in <top triangle> must contain the same number of E's as <side 1> in <bottom triangle> and similarly for <side 2>). This is a constraint which must be placed on the grammar in some way before it can be used. It should be noted that there are only two limitations on the size of a

rectangle: (1) It can only be as small as the arm lengths on the right angles  and (2) It can be as large as the generation area (and other constraints, if any) will allow.

EXAMPLE:

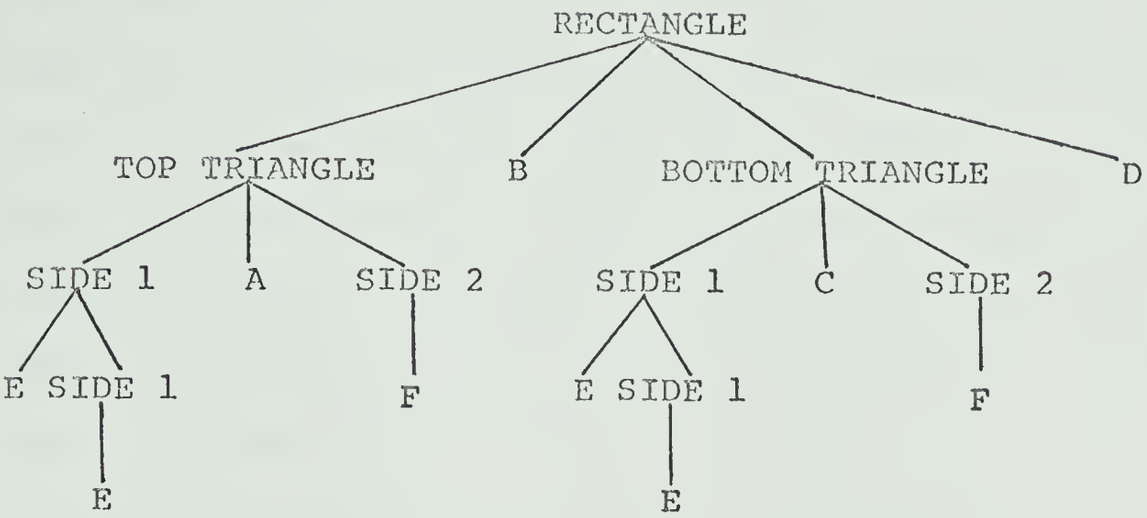


FIGURE 7.2

We now have the question: Will this method help us to improve the picture definition facilities of graphics software? In the definition of the class of rectangles using GRIDSUB, it was necessary to put certain conditions on the values of X_i and Y_i so that the angles were right angles and the opposite sides were equal. Using Ledley's grammar, the right angles are explicitly defined but it is still necessary to place certain constraints on the definition

so that the opposite sides are equal.

In the GRIDSUB definition, there was only one primitive - the line segment. There were two implicit parts of the definition: The lines were joined at right angles and opposite sides were the same length. In this definition there are essentially two primitives (although we named six), the line segment and the right angle. There is, however, only one implicit part of the definition: Opposite sides are the same length. Obviously, in this definition fewer factors are implicit than in the definition in GRIDSUB; however, the explicit part of the definition is more complicated.

It appears that Ledley's method does not suggest any changes that could be made in order to improve the existing graphics facilities.

7.3 Definitions Based on Narasimhan's Methods:

Narasimhan developed a syntactic scheme for describing pictures. We looked at two applications using these methods and will now describe the class of rectangles employing the procedures used in each of the two applications.

7.3.1 Narasimhan's Method for Describing Bubble Chamber Pictures:

The processing of a bubble chamber picture involves isolating the black points, joining these points into lines and curves, finding the points of intersection, and hence

identifying and describing the picture. The problem (describing the class of rectangles - for generation in particular) for which we want to use the description is the reverse of Narasimhan's problem. Hence, we will have to provide the description of the class of pictures in a different form.

Obviously, it is quite simple to describe a rectangle using points as the machine primitives. This could be done by using a matrix of points where a given point is either blank or non-blank according to a fixed definition (See Figure 7.3). Another possible description is a sequence of points with each point having a relative location (position) with respect to the points surrounding it.

```
bbbbbbbbbb
bb5EEEE5bb
bbNbbbbNbb
bbNbbbbNbb
bbNbbbbNbb
bbNbbbbNbb
bb5EEEE5bb
bbbbbbbbbb
```

Figure 7.3

Now consider the LINE or VECTOR as the primitive of this system instead of the POINT. The intersections or vertices will have certain attributes which are functions of the attributes of the VECTOR. This system (whether POINTS or VECTORS are the primitive of the system) involves a hierarchy of object types and, therefore, the object (to be analyzed) has certain attributes which are actually functions

of the attributes of the system primitive.

EVENT (attributes $a_1, \ldots, a_n$)

↑

$a_i = f_i(w_1, \ldots, w_n)$

VECTOR (attributes $w_1, \ldots, w_n$)

When a hierarchical description system is to be used for generation (or synthesis) rather than analysis, the functions are applied in the reverse order (i.e., the attributes of the primitives are functions of the attributes of the object types at a higher level). This process is similar to that used in existing graphics systems, like GRIDSUB, for design problems, like CALD.

BLOCK (attributes $a_1, \ldots, a_n$)

$w_i = f_i(a_1, \ldots, a_n)$

↓

VECTOR (attributes $w_1, \ldots, w_n$)

For example, if one attribute of a primitive is position, the value of this attribute is undefined until the position of the picture has been specified. Hence, for a rectangle, the position of any one vector in the rectangle is a function of the origin of the complete rectangle. Therefore, the position attribute of the primitive is a function of the position attribute of the picture.

The important thing to notice is that this description of the class of rectangles would allow the programmer to specify size, position, orientation, etc. at

generation time (that is, display time) rather than at the time of definition.

7.3.2 Generating English Letters:

In order to use a grammar like the one used for generating English letters, it is necessary to define two primitives or, rather, two classes of primitives: A horizontal line, called A, and a vertical line, called B. These must be classes of primitives so that the lengths of A and B can be changed for the generation of a given rectangle. The primitives A and B each have two vertices (points of connection) as shown in Figure 7.4. A grammar for describing rectangles might be like the following:

rectangle → tarm.barm (11,22)

tarm (1,2) → A.B (21;1;2)

barm (1,2) → B.A (21;1;2)

An example of the use of the grammar is shown by the tree structure in Figure 7.5.

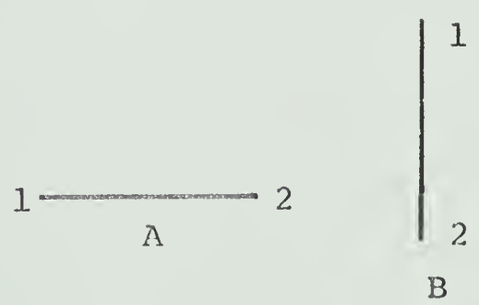


FIGURE 7.4

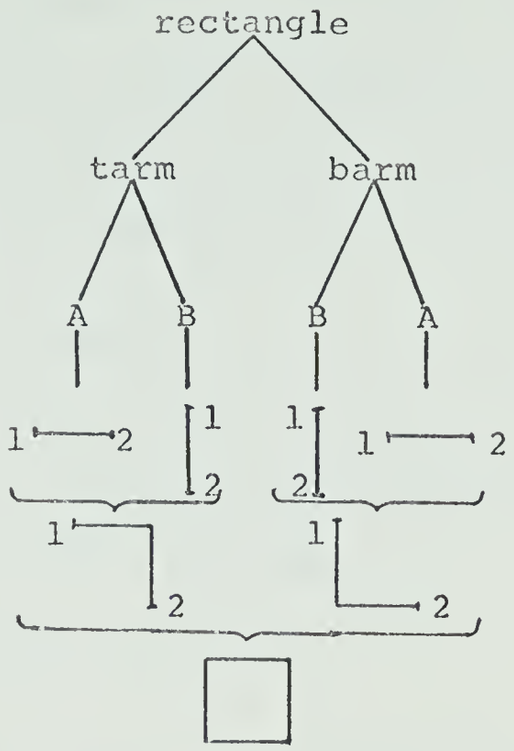


FIGURE 7.5

The method for the generation of English letters uses a higher level primitive than the point (i.e., the line segment), making the definition of the class of rectangles much easier to state, but it still retains the feature of attributes. These attributes only indicate the height and the thickness of the primitives. It would seem possible to extend this attribute list to include other values of a primitive such as position and angle of rotation. It should also be noted that the list indicating what and where vertices are joined is similar to an attribute list. It also specifies certain information, although this information may describe a picture at a higher level than the problem primitive. This list is important in that Narasimhan uses it to cope with the problem of there being a number of possible joining points on a primitive or in higher level

pictures. It is possible that this list could contain, besides joining points, other attributes, such as location, which describe the complete picture (as distinct from the attribute list for the problem primitives).

It would seem that a graphics system based on Narasimhan's work would allow pictures to be described more explicitly in terms of the problem primitives than is the case in existing graphics software.

7.4 Description Using PDL:

The first step in using PDL is to define the necessary primitive classes. To define rectangles, there must be such classes A and B, respectively the class of horizontal lines and the class of vertical lines (see Figure 7.6). The attribute list for each of the primitives is:

$$A = (A, X_1, X_2, Y)$$

$$B = (B, Y_1, Y_2, X)$$

Using the PDL, we can describe the class of rectangles by the following statement:

rectangle → ((A + B) * (B + A)).

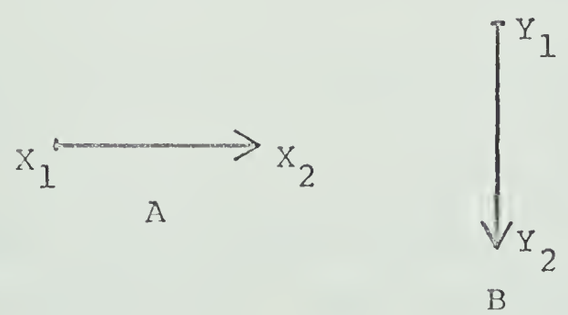


FIGURE 7.6

Before generation of a rectangle it will be necessary to describe a particular A and B. For each A, X_1 and X_2 are the same but Y will be Y_1 for one A and Y_2 for the other A. Similarly, for each B, Y_1 and Y_2 are the same but X will be X_1 for one B and X_2 for the other B. Figure 7.7 shows the generation of a rectangle.

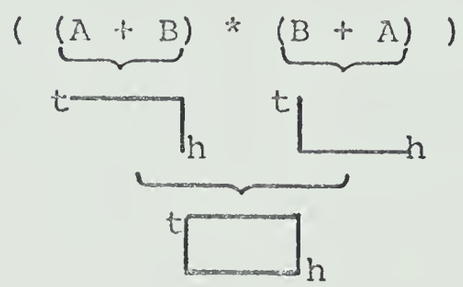


FIGURE 7.7

Obviously, the description of the class of rectangles is very similar to that using Narasimhan's method for generating English letters, but with a number of differences: First, these primitives may have only two vertices while Narasimhan's may have any number. Second, these attribute lists may specify any variables while Narasimhan's seem to be limited. Third, Narasimhan's grammar only joins two objects in one statement while one PDL statement may join any number of objects.

The fact that any picture, primitive or complete, has only two vertices will cause some difficulties. For example, consider the AND-gate primitive of the CALD system. (See Figure 7.8). This primitive may be joined to

other primitives at points A, B and C. Obviously, the primitive cannot be specified with three vertices; however, it seems to be almost impossible to describe the AND-gate with only two vertices. No doubt there are other classes of pictures which require more than two vertices if they are to be described in a reasonable and convenient manner.

FIGURE 7.8

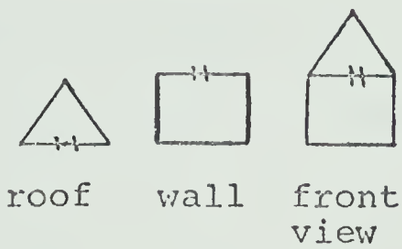


FIGURE 7.9

The fact that a primitive defined for a PDL system may have, theoretically, an unlimited list of attributes is one advantage of this method. For example, consider a class of pictures like Ledley's houses with primitives as shown in Figure 7.9. It would be possible to give the length of the marked lines in the attribute list and to compare for the same lengths before constructing the front view of a house. In this way, we would be able to overcome the problem of size compatibility (i.e., in this case, the length of the roof base must be equal to the length of the top of the wall) and perhaps other problems such as differences in angles of rotation.

7.5 Conclusion:

Ledley's method does not seem to suggest any possible improvements which might be made to graphics

software. On the other hand, while the other methods discussed present some problems, they do contain certain concepts that might be used to design better graphics software. Probably the most important of these was the idea of an attribute list, particularly when used in a hierarchical system.

In the next chapter, we shall consider some changes which might be made to existing software such as GRIDSUB. It is to be hoped that these changes will lead to better graphics software and possibly to a more formal graphics system which allows explicit definition of classes of pictures.

CHAPTER 8 - IMPROVEMENTS TO GRAPHICS SOFTWARE

8.1 Introduction:

In the preceding chapters, we have looked at various methods using linguistic techniques for describing classes of pictures, and the way in which each method might be used to describe a particular class of pictures. The question we shall consider now is: What have we learned that might help us build better graphics systems? Of course, it is important to realize that graphical communication is only one part (although an important part) of computer-aided design. Before we consider this question, we shall discuss the structure of such a system.

8.2 Structure of Computer-Aided Design Systems:

Computer-aided design (CAD) has been defined as "the adaptation of the computer for automated industrial, biological, statistical, (etc.) design through visual devices" (Sippl, 1966). It is reasonable to expect that a CAD system should allow the user to solve his design problem faster than if he were not using a computer. If this is not true then the computer does not have any advantage for the designer. In general, the systems developed to date require a good deal of programmer development and are built for only one particular design problem and hence are not adaptable to other design problems. In other words, the amount of genuine computer-aided design remains small even though new

systems are being built. Therefore, we need to produce better software if we wish to improve computer-aided design.

What we are interested in finding are better ways of defining drawings at one level of the hierarchy (see Chapter 6) in terms of drawings at lower levels of the hierarchy so that computer-aided design systems require less complex programming and are more adaptable to new situations. In the next section, we shall consider some changes which might be incorporated into GRIDSUB.

8.3 Definition of Display Primitives:

At present, the problem primitives of the CALD system are defined explicitly in terms of the machine primitives. The NOT-gate, for example, has a definition which looks like:

```
CALL BLOCK (1)
CALL VECTOR (A,X1,Y1,NB,ND)
CALL VECTOR (A,X2,Y2,NB,ND)
CALL VECTOR (A,X3,Y3,NB,ND)
CALL VECTOR (A,X4,Y4,NB,ND)
CALL MOVE (A,X3,Y3)
CALL VECTOR (A,X5,Y5,NB,ND)
CALL VECTOR (A,X6,Y6,NB,ND)
CALL ENDBLK
```

The gate is displayed by the following instruction:

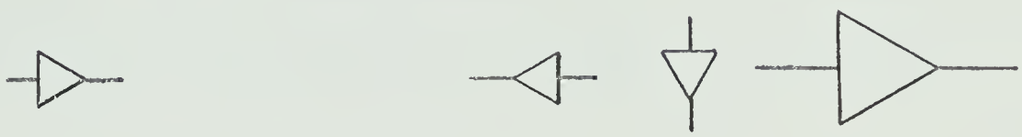
```
CALL DISPLAY (1,1,Z1,Z2).
```

The variables, X_i and Y_i, are undefined when the definition of the gate is made. When the gate is displayed at a certain position, say (Z_1,Z_2), the variables X_i and Y_i are functions

of Z_1 and Z_2:

$$X_i = f_i(Z_1) \text{ and } Y_i = g_i(Z_2)$$

In defining the NOT-gate we are able to get only one shape (Figure 8.1a) which may be displayed at any position. We would also like to be able to have size and orientation as functions of the display (Figure 8.1b). This would actually allow us to define classes of primitives (as in Miller and Shaw's PDL) instead of defining each primitive individually.



a b

FIGURE 8.1

When the user of a CAD system is drawing pictures, he is not interested in (and, in fact, does not need to know about) the display primitives. The lowest level in his systems hierarchy is the problem primitive. If we look at our problem from his viewpoint, the obvious solution is to have a display command like:

$$\text{DISPLAY } (Z_1, Z_2, Z_3, \ldots, Z_n)$$

where one variable indicates scale, another indicates orientation and so on. This is like Narasimhan's idea of having attribute lists.

A system like GRIDSUB does not have the facilities for handling a display function like this one. It would seem that there are three possible solutions: (1) Redefine the display function; (2) Redefine the method of BLOCK definition; or (3) Redefine the display primitives or introduce new ones.

Let us look at the NOT-gate used in the CALD system and consider the possibilities of redesigning some part of its definition so that we can specify other attributes besides location. The display command that the builder would like to provide for the user of the design system could look like:

CALL DISPLAY (X, Y; N; M^{O})

where X,Y is location, N is scale, M^{O} is orientation and where the basic NOT-gate would look like:

(X,Y)

scale = 1
orientation = 0^{O}.

Suppose we consider the vector to be a display primitive, but allow it to be defined in a way similar to that in which primitives were defined in Miller and Shaw's work. The class of vectors could then look something like:

VECTOR = (head function, tail function, line function, size function, direction function).

The head and tail values would be specified in terms of the tail position of the previous vector, the scale value, the orientation and a function definition of the line.

We shall describe how this might work using the NOT-gate as an example. In Figure 8.2a the vectors are numbered in the order in which they will be defined, and the relative lengths are shown in Figure 8.2b.

FIGURE 8.2

The vector definitions would be:

(X ,Y ; X_1=X+2,Y_1=Y; X_1=X+(2*N); f(m))

(X_1,Y_1; X_2=X_1,Y_2= Y_1+1; Y_2=Y_1+(1*N); f(m))

(X_2,Y_2; X_3=X_2+$\sqrt{8}$,Y_3=Y; X_3=X_2+($\sqrt{8}$*N); f(m))

(X_3,Y_3; X_4=X_3+2,Y_4=Y_3; X_4=X_3+(2*N); f(m))

(X_3,Y_3; X_5=X_1,Y_5=Y-1; Y_5=Y-(1*N); f(m))

(X_5,Y_5; X_1,Y_1; ; f(m))

The first pair of values (attributes) indicates the position of one end of the vector; the second pair of values (attributes) indicates the position of the other end and are specified in terms of previously defined values assuming scale is 1 and orientation is 0^o. If scale is not 1, then the third attribute indicates the changes which must be made (this could be incorporated into the second attribute) in order to get the proper end points. The fourth attribute is a function which will rotate the vector depending upon

the orientation specified. This function could be applied to each vector as it is entered in the display file or to the part of the display file which describes the defined problem primitive. Also, the gate could be rotated around the X,Y position or it could be rotated around a specified center point.

It is likely that there would be different ways to specify the display primitives so that the problem primitives may be constructed as classes of primitives with a number of attributes. It should be noted that most (if not all) straight line drawings may be constructed from line segments which are joined at either of the ends but not in the middle. Therefore, there is no need to introduce display primitives (as done by Narasimhan) which have more than two points of connection.

8.4 Definition of Pictures:

We now have a method for defining and displaying classes of primitives. This method will give us greater flexibility in defining the problem primitives which the user of the system needs. However, we have not discussed any way to designate which points of a primitive may be joined to other primitives and what limitations must be placed on the primitives which can be joined.

One possible solution is to introduce a notation like Narasimhan's where the vertices were numbered and then the syntax indicated which vertices could be joined. If

we go back to our definition of the NOT-gate, we see that it has two points of connection which are at X,Y and at X_4,Y_4. The other gates, AND and OR, each have three points of connection - two for input and one for output. Therefore, we might name the points INPUT1, INPUT2 and OUTPUT or just number the vertices 1, 2 and 3 as shown in Figure 8.3. Suppose we assume that these points of connection can be indicated in some convenient way so that for each primitive being displayed the points are readily available.

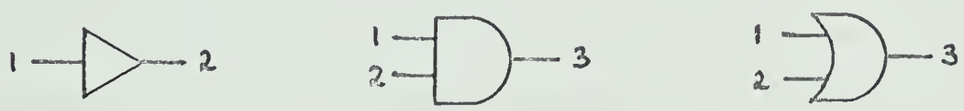


FIGURE 8.3

Now that we can define classes of primitives with specified connecting points, we must consider another question: Can we introduce a grammar that may be used to define classes of pictures explicitly rather than implicitly? After some consideration, it appears that we cannot yet devise a grammar or combination of grammars that could be used to describe the class of circuit diagrams. This inability to define the class of circuit diagrams probably means that the grammars studied are not powerful enough to describe pictures of such complexity. Complications arise because:

(1) The number of unattached nodes in a subcircuit of a given number of primitives may not be the same as the number of unattached nodes in another subcircuit having the

same number of primitives. Also, when a primitive is joined to a subcircuit, the number of unattached nodes added to the subcircuit depends on the type of primitive. Therefore, it is almost impossible to use a method like the one Narasimhan uses for English letters to define explicitly this class of pictures.

(2) Certain primitives have more than two points of connection. This precludes the use of Miller and Shaw's PDL to describe the class of pictures.

(3) When we consider circuit diagrams, we are not interested in a class of pictures but really we are looking at a (theoretically) infinite number of classes of pictures. (This is because there is an infinite number of combinations of inputs and outputs for circuit diagrams.) If we were to consider some restricted class of circuit diagrams, it might be possible to define that class explicitly.

It appears that we are not yet able to define a grammar that will explicitly describe classes of pictures as complex as logic circuit diagrams.

CHAPTER 9 - CONCLUDING REMARKS

In this thesis we have studied ways in which linguistic techniques, devised for defining string languages, may be extended to define classes of drawings using picture grammars. An aim of this work was to suggest improvements to graphics software facilities, and perhaps to find ways to build more formal software for a computer graphics system. The results of this work were partially successful in fulfilling this aim.

New ways were suggested for specifying the machine primitives, the important difference being that certain functions (used to generate a particular machine primitive) could be specified at display time. Also, another way of defining and displaying the BLOCK (or problem primitive) used in GRIDSUB was suggested. This method allowed the definition of classes of problem primitives, and is more general and therefore potentially more useful. These results show that the study of picture grammars can produce insight into the problem of building both better and more formal software.

It was felt that this work might also lead to methods of obtaining better overall organization of the hierarchy of picture description in, for example, a computer-aided design system. It would have been desirable to build a picture grammar, based on the methods studied, which could

describe classes of fairly complex pictures explicitly. Unfortunately, this area of study has not, as yet, yielded any practical methods. Therefore, it is desirable that the study of generative grammars be continued. One possibility might be to define a grammar based on PDL but with primitives having more than two points of connection.

If it is assumed that a generative grammar, which meets the requirements, can be built to define a given class of pictures, then there are other questions that might be considered. One question is: Is it possible to build an "error free" system? An "error free" system could be defined as a CAD system which will "generate" (i.e., develop in response to requests from the user) all pictures of the class and will not generate any picture which is not in the class. The second question is: If the system is given the location where a primitive is to be placed, can the system display that primitive with the correct size and orientation relative to the other parts of the picture which have already been displayed?

There is one other question which relates to the assumption that a generative grammar can be built to define a given class of pictures. If a class of pictures can be explicitly defined using a generative grammar, can a general purpose processor be built which will use the generative grammar description (for any class of pictures) to generate the class of pictures for a particular design problem? Any

consideration of this question would probably involve a study of techniques developed for "syntax-directed" translators for string languages.

This work has barely begun to study the possible uses of generative picture grammars in problems involving computer graphics. However, it has shown that possibilities for applying these techniques within computer graphics systems do exist and that this research should be continued.

REFERENCES

1. Boehm, B. W., V. R. Lamb, R. L. Mobley, and J. E. Rieber, "POGO: Programmer-Oriented Graphics Operation", Proceedings of AFIPS Spring Joint Computer Conference, 1969.

2. Cancro, Robert and D. L. Slotnick, "Some Thoughts on Displays", Emerging Concepts in Computer Graphics, Edited by Don Secrest and Jurg Nievergelt, W. A. Benjamin, Inc., 1968.

3. Deecker G. F. P., "Computer Graphics and Campus Planning", M.Sc. Thesis, Department of Computing Science, University of Alberta, Fall 1970.

4. Fetter, William A., "Computer Graphics", Emerging Concepts in Computer Graphics, Edited by Don Secrest and Jurg Nievergelt, W. A. Benjamin, Inc., 1968.

5. Gray, J. C., "Compound Data Structure for Computer Aided Design: A Survey", Proceedings A.C.M. National Meeting, 1967.

6. Huen, W. H., F. B. Jacobsen, K. F. May, and J. P. Penny, "Reference Manual: Computer Graphics for the Fortran Programmer", Computing Center Publication, University of Alberta, January 1970.

7. Ledley, Robert S., "High-Speed Automatic Analysis of Biomedical Pictures", Science, Vol. 146, 1964.

8. Ledley, Robert S., Programming and Utilizing Digital Computers, McGraw-Hill Book Company Inc., 1962.

9. Ledley, R. S., L. S. Rotolo, T. J. Golab, J. D. Jacobsen, M. D. Ginsberg and J. B. Wilson, "FIDAC: Film Input to Digital Automatic Computer and Associated Syntax-Directed Pattern-Recognition Programming System", Optical Processing, The Massachusetts Institute of Technology Press, 1964.

10. Narasimhan, R., "Labeling Schemata and Syntactic Descriptions of Pictures", Information and Control, Vol. 7, 1964.

11. Narasimhan, R., "Syntax-Directed Interpretation of Classes of Pictures", Communications of the A.C.M., Vol. 9, No. 3, March 1966.

12. Narasimhan, R. and V. S. N. Reddy, "A Generative Model for Handprinted English Letters and Its Computer Implementation", ICC Bulletin, Vol. 6, No. 4, 1967.

13. Shaw, Alan C., "A Proposed Language for the Formal Description of Pictures", Stanford Linear Accelerator Center, Stanford University, GSG Memo 28, February 1967.

14. Sippl, Charles J., Computer Dictionary and Handbook, Howard W. Sans and Co., Inc., 1966.

BIBLIOGRAPHY

1. Clowes, M. B., "A Generative Picture Grammar", Computing Research Section, Commonwealth Scientific and Industrial Research Organization, Seminar Paper No. 6, April 1967.

2. Clowes, M. B., "Perception, Picture Processing and Computers", Machine Intelligence 1 (Chapter 12), Edited by N. L. Collins and Donald Miche, American Elsevier Publishing Company, Inc., 1967.

3. George, Jim, "Picture Generation Using the Picture Calculus", Stanford Linear Accelerator Center, Stanford University, GSG 50, December 1967.

4. Greancas, E. C., R. F. Menghen, R. J. Norman, and P. Easinger, "The Recognition of Handwritten Numerals by Contour Analysis", IBM Journal of Research and Development, Vol. 7, No. 1, January 1963.

5. Hayashi, Hideyuki, Sheila Duncan, S. Kuno, "Graphical Input/Output of Nonstandard Characters", Communications of the A.C.M., Vol. 11, No. 9, September 1968.

6. Kamentsky, L. A., and C. N. Liu, "Computer-Automated Design of Multifont Print Recognition Logic", IBM Journal of Research and Development, Vol. 7, No. 1, January 1963.

7. Kirsch, Russell A., "Computer Interpretation of English Text and Picture Patterns", IEEE Transactions on Computers, Vol. C-13, No. 4, August 1964.

8. Kuhl, Frank, "Classification and Recognition of Hand-Printed Characters", IEEE International Convention Record, Part 4, 1963.

9. Ledley, Robert Steven, Use of Computers in Biology and Medicine, McGraw-Hill Book Company, 1965.

10. Ledley, R. S., J. Jacobsen, and M. Belson, "BUGSYS: A Programming System for Picture Processing - Not for Debugging", Communications of the A.C.M., Vol. 9, No. 2, February 1966.

11. Ledley, R. S., L. S. Rotolo, M. Belson, J. Jacobsen, J. Wilson, and T. Golab, "Pattern Recognition Studies in the Biomedical Sciences", Proceedings of the Spring Joint Computer Conference, 1966.

12. Ledley, Robert S., and James B. Wilson, "Automatic-Programming-Language Translation Through Syntactical Analysis", Communications of the A.C.M., Vol. 5, 1962.

13. Miller, W. F., and A. C. Shaw, "A Picture Calculus", Stanford Linear Accelerator Center, Stanford University, GSG 40, June 21, 1967.

14. Narasimhan R., "Intelligence and Artifical Intelligence", Tata Institute of Fundamental Research, Computer Group - Research and Development, Technical Report No. 16, October 1966.

15. Narasimhan R., "Programming Languages and Computers: A Unified Metatheory", Advances in Computers, Vol. 8, 1967.

16. Noyelle, Yves, "Implementation on the PDP-1 of a Subset of the Picture Calculus", Stanford Linear Accelerator Center, Stanford University, CGTM 31, November 1967.

17. Shaw, Alan Cary, "The Formal Description and Parsing of Pictures", Ph.D. Thesis, Stanford University, April 1968.

18. Shaw, Alan C., "A Picture Calculus - Further Definitions and Some Basic Theorems", Stanford Linear Accelerator Center, Stanford University, GSG Memo 46, June 1967.

19. Simmons, R. F., "Answering English Questions By Computer: A Survey", Communications of the A.C.M., Vol. 8, No. 1, January 1965.

20. Simmons, Robert F. and David L. Londe, "Namer: A Pattern Recognition System for Generating Sentences about Relations Between Line Drawings", Technical Memorandum, System Development Corporation, June 5, 1964.

21. van der Lans, J., "Hummingbird, Automatic Film Digitizers at the Stanford Linear Accelerator Center", Stanford Linear Accelerator Center, Stanford University, SLAC-PUB-251, January 1967.

22. Wolman, Barry L., "Operators for Manipulating Language Structures", Massachusetts Institute of Technology Electronic Systems Laboratory, Cambridge, Massachusetts, March 18, 1966.

APPENDIX I

Linguistics may be defined as the study of the structure and development of a language and its relationship to other languages. In this work, the term LINGUISTIC TECHNIQUES refers to the various methods which the linguist uses to define the structure of the language. The technique of most interest is the Linguistic Grammar.

A LINGUISTIC GRAMMAR is a description of the structure of a language. That is, it describes valid sentences in the language and gives a structured description of the sentence. The linguistic grammar which is important, in this work, is the Generative Grammar.

A GENERATIVE GRAMMAR describes the structure of a language using a set of production rules which are applied in a particular sequence to produce a sentence in the language. A generative grammar is context sensitive if a production rule can apply to a particular element only if the element is in the specified context. If a production rule may apply regardless of context, a grammar is context free. The context free grammar is of more interest in the study of graphical systems than the context sensitive grammar.

A PICTURE GRAMMAR is analogous to a generative grammar in that it describes structure. The difference is that the production rules describe well-formed classes of pictures instead of languages. The ideal picture grammar

would be context free, but most are context sensitive in the sense that some part of the description must be implicitly part of the description rather than being explicit in the production rules.